```
# Import the modules
import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.metrics import balanced_accuracy_score, confusion_matrix, classification_report
```

Split the Data into Training and Testing Sets

Step 1: Read the `lending_data.csv` data from the `Resources` folder into a Pandas DataFrame.

```
# Read the CSV file from the Resources folder into a Pandas DataFrame
lendingDataDF = pd.read_csv("lending_data.csv")

# Review the DataFrame
lendingDataDF.head()
```

	loan_size	interest_rate	borrower_income	debt_to_income	num_of_accounts	derogatory_marks	total_debt	loan_status
0	10700.0	7.672	52800	0.431818	5	1	22800	0
1	8400.0	6.692	43600	0.311927	3	0	13600	0
2	9000.0	6.963	46100	0.349241	3	0	16100	0
3	10700.0	7.664	52700	0.430740	5	1	22700	0
4	10800.0	7.698	53000	0.433962	5	1	23000	0

Next steps: Generate code with `lendingDataDF` View recommended plots New interactive sheet

Step 2: Create the labels set (`y`) from the "loan_status" column, and then create the features (`X`) DataFrame from the remaining columns.

```
# Separate the data into labels and features

# Separate the y variable, the labels
y = lendingDataDF['loan_status']

# Separate the X variable, the features
X = lendingDataDF.drop(columns=['loan_status'])
```

```
# Review the y variable Series
# Print the first few entries of y
print("First few entries of y:")
print(y.head())

# Print summary statistics for y
print("\nSummary statistics of y:")
print(y.describe())

# Print the distribution of loan_status values
print("\nDistribution of loan_status values:")
print(y.value_counts())
```

```
First few entries of y:
0    0
1    0
2    0
3    0
4    0
```

```
Name: loan_status, dtype: int64

Summary statistics of y:
count    77536.000000
mean         0.032243
std          0.176646
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: loan_status, dtype: float64

Distribution of loan_status values:
loan_status
0    75036
1     2500
Name: count, dtype: int64
```

```
# Review the X variable DataFrame
# Print the first few rows of X
print("First few rows of X:")
print(X.head())

# Print summary statistics of X
print("\nSummary statistics of X:")
print(X.describe())

# Print information about the DataFrame, including column names and data types
print("\nInformation on X:")
print(X.info())
```

```
First few rows of X:
   loan_size  interest_rate  borrower_income  debt_to_income  num_of_accounts  \
0    10700.0          7.672            52800        0.431818                5
1     8400.0          6.692            43600        0.311927                3
2     9000.0          6.963            46100        0.349241                3
3    10700.0          7.664            52700        0.430740                5
4    10800.0          7.698            53000        0.433962                5

   derogatory_marks  total_debt
0                 1       22800
1                 0       13600
2                 0       16100
3                 1       22700
4                 1       23000

Summary statistics of X:
          loan_size  interest_rate  borrower_income  debt_to_income  \
count  77536.000000   77536.000000     77536.000000    77536.000000
mean    9805.562577       7.292333     49221.949804        0.377318
std     2093.223153       0.889495      8371.635077        0.081519
min     5000.000000       5.250000     30000.000000        0.000000
25%     8700.000000       6.825000     44800.000000        0.330357
50%     9500.000000       7.172000     48100.000000        0.376299
75%    10400.000000       7.528000     51400.000000        0.416342
max    23800.000000      13.235000    105200.000000        0.714829

       num_of_accounts  derogatory_marks    total_debt
count     77536.000000      77536.000000  77536.000000
mean          3.826610          0.392308  19221.949804
std           1.904426          0.582086   8371.635077
min           0.000000          0.000000      0.000000
25%           3.000000          0.000000  14800.000000
50%           4.000000          0.000000  18100.000000
75%           4.000000          1.000000  21400.000000
max          16.000000          3.000000  75200.000000

Information on X:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77536 entries, 0 to 77535
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   loan_size         77536 non-null  float64
```

```
 1   interest_rate     77536 non-null  float64
 2   borrower_income   77536 non-null  int64
 3   debt_to_income    77536 non-null  float64
 4   num_of_accounts   77536 non-null  int64
 5   derogatory_marks  77536 non-null  int64
 6   total_debt        77536 non-null  int64
dtypes: float64(3), int64(4)
memory usage: 4.1 MB
None
```

Step 3: Check the balance of the labels variable (`y`) by using the `value_counts` function.

```
# Check the balance of our target values
print("Value counts for loan_status:")
print(y.value_counts())
```

```
Value counts for loan_status:
loan_status
0    75036
1     2500
Name: count, dtype: int64
```

Step 4: Split the data into training and testing datasets by using `train_test_split`.

```
# Import the train_test_learn module
from sklearn.model_selection import train_test_split

# Split the data using train_test_split
# Assign a random_state of 1 to the function
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

# Review the shapes of the resulting datasets
print("Training features shape:", X_train.shape)
print("Testing features shape:", X_test.shape)
print("Training labels shape:", y_train.shape)
print("Testing labels shape:", y_test.shape)
```

```
Training features shape: (58152, 7)
Testing features shape: (19384, 7)
Training labels shape: (58152,)
Testing labels shape: (19384,)
```

Create a Logistic Regression Model with the Original Data

Step 1: Fit a logistic regression model by using the training data (`X_train` and `y_train`).

```
# Import the LogisticRegression module from SKLearn
from sklearn.linear_model import LogisticRegression

# Instantiate the Logistic Regression model
# Assign a random_state parameter of 1 to the model
logistic_model = LogisticRegression(random_state=1)

# Fit the model using training data
logistic_model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=1)
```

Step 2: Save the predictions on the testing data labels by using the testing feature data (`X_test`) and the fitted model.

```
# Make a prediction using the testing data
predictions = logistic_model.predict(X_test)

# Display the first few predictions
print("Predictions on the testing data:")
print(predictions[:5])
```

```
Predictions on the testing data:
[0 0 0 0 0]
```

Step 3: Evaluate the model's performance by doing the following:

- Calculate the accuracy score of the model.
- Generate a confusion matrix.
- Print the classification report.

```
# Print the balanced_accuracy score of the model
balanced_acc = balanced_accuracy_score(y_test, predictions)
print("Balanced Accuracy Score:", balanced_acc)
```

```
Balanced Accuracy Score: 0.967989851522121
```

```
# Generate a confusion matrix for the model
conf_matrix = confusion_matrix(y_test, predictions)
print("\nConfusion Matrix:\n", conf_matrix)
```

```

Confusion Matrix:
 [[18655   110]
 [   36   583]]
```

```
# Print the classification report for the model
class_report = classification_report(y_test, predictions)
print("\nClassification Report:\n", class_report)
```

```

Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.99      1.00     18765
           1       0.84      0.94      0.89       619

    accuracy                           0.99     19384
   macro avg       0.92      0.97      0.94     19384
weighted avg       0.99      0.99      0.99     19384
```

Step 4: Answer the following question.

Question: How well does the logistic regression model predict both the `0` (healthy loan) and `1` (high-risk loan) labels?

Answer: The model does a good job correctly identifying healthy loans (0), but it has more difficulty accurately predicting high-risk loans (1). This means while the overall accuracy is decent, there's room to improve its detection of risky loans.

Predict a Logistic Regression Model with Resampled Training Data

Step 1: Use the `RandomOverSampler` module from the imbalanced-learn library to resample the data. Be sure to confirm that the labels have an equal number of data points.

```
# Import the RandomOverSampler module form imbalanced-learn
from imblearn.over_sampling import RandomOverSampler

# Instantiate the random oversampler model
# # Assign a random_state parameter of 1 to the model
ros = RandomOverSampler(random_state=1)

# Fit the original training data to the random_oversampler model
X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)

# Confirm the labels have an equal number of data points
print("Resampled target value counts:")
print(y_train_resampled.value_counts())
```

```
Resampled target value counts:
loan_status
0    56271
1    56271
Name: count, dtype: int64
```

```
# Count the distinct values of the resampled labels data
resampled_counts = y_train_resampled.value_counts()
print("Resampled labels counts:")
print(resampled_counts)
```

```
Resampled labels counts:
loan_status
0    56271
1    56271
Name: count, dtype: int64
```

Step 2: Use the `LogisticRegression` classifier and the resampled data to fit the model and make predictions.

```
# Instantiate the Logistic Regression model
# Assign a random_state parameter of 1 to the model
logistic_model_resampled = LogisticRegression(random_state=1)

# Fit the model using the resampled training data
logistic_model_resampled.fit(X_train_resampled, y_train_resampled)

# Make a prediction using the testing data
predictions_resampled = logistic_model_resampled.predict(X_test)

# display the first few predictions
print("Predictions on the testing data with the resampled model:")
print(predictions_resampled[:5])
```

```
Predictions on the testing data with the resampled model:
[0 0 0 0 0]
```

Step 3: Evaluate the model’s performance by doing the following:

- Calculate the accuracy score of the model.
- Generate a confusion matrix.
- Print the classification report.

```
# Print the balanced_accuracy score of the model
balanced_acc_resampled = balanced_accuracy_score(y_test, predictions_resampled)
print("Balanced Accuracy Score (Resampled):", balanced_acc_resampled)
```

```
Balanced Accuracy Score (Resampled): 0.9935981855334257
```

```
# Generate a confusion matrix for the model
conf_matrix_resampled = confusion_matrix(y_test, predictions_resampled)
print("\nConfusion Matrix (Resampled):\n", conf_matrix_resampled)
```

```
Confusion Matrix (Resampled):
 [[18646   119]
 [    4   615]]
```

```
# Print the classification report for the model
class_report_resampled = classification_report(y_test, predictions_resampled)
print("\nClassification Report (Resampled):\n", class_report_resampled)
```

```
Classification Report (Resampled):
               precision    recall  f1-score   support

           0       1.00      0.99      1.00     18765
           1       0.84      0.99      0.91       619

    accuracy                           0.99     19384
   macro avg       0.92      0.99      0.95     19384
weighted avg       0.99      0.99      0.99     19384
```

Step 4: Answer the following question

Question: How well does the logistic regression model, fit with oversampled data, predict both the `0` (healthy loan) and `1` (high-risk loan) labels?

Answer: With oversampling, the model predicts both healthy loans (0) and high-risk loans (1) more evenly. It shows improved detection of high-risk loans while still accurately identifying healthy loans, leading to a more balanced overall performance.